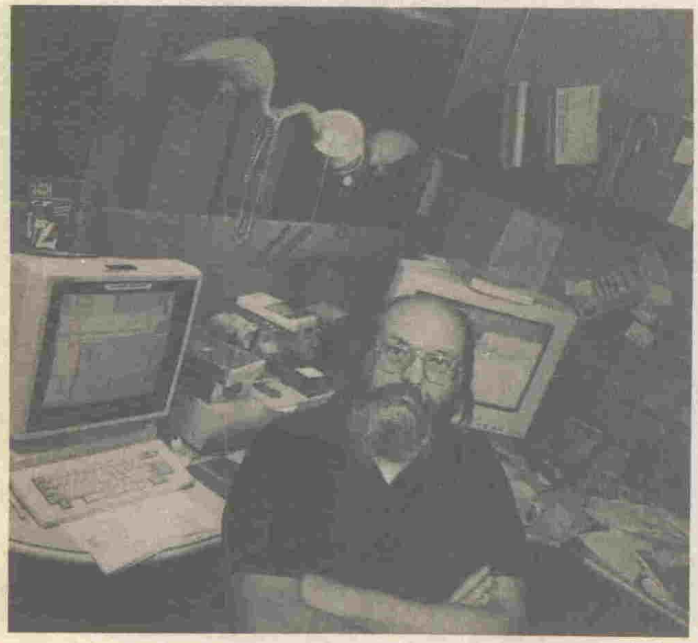
สุทธิศักดิ์ พงศ์ธนาพาณิช

Advanced BASIC Programming Technique : เทคนิคการออปติไมซ์ Visual Basic 5.0 #1



แนะนำวิธีการออปติไมซ์ซอร์สโค้ดของ Visual Basic 5.0 ซึ่งสามารถช่วยเพิ่มความเร็วของโปรแกรมได้มาก

จากบทความตอนที่แล้วผู้เขียนได้แนะนำให้ผู้อ่านรู้จักกับความสามารถด้าน OLE DnD ของ Visual Basic 5.0 ซึ่งเป็นคุณสมบัติที่ได้รับการพัฒนาจนเทียบเท่ากับสิ่งที่มีใน Visual C++ ในการเข้าถึงสถาปัตยกรรม OLE ของวินโดวส์ 32 บิต และก่อนที่ผู้เขียนจะแนะนำให้ผู้อ่านทราบถึงความสามารถใหม่ ๆ ของ Visual Basic 5.0 ไม่ว่าจะเป็นการสร้าง OLE.DLL, OLE.EXE หรือ ActiveX.OCX เป็นต้น ผู้เขียนก็จะขอแนะนำวิธีการออปติไมซ์ซอร์สโค้ดของ Visual Basic 5.0 เสียก่อน เพื่อให้การติดตามบทความตอนต่อ ๆ ไป ผู้อ่านจะได้ไม่สงสัยว่าทำไมโค้ดบางส่วนจึงได้ถูกลดรูปหรือเปลี่ยนแปลงรูปแบบในการเขียนโค้ดในระดับโครงสร้าง ซึ่งการออปติไมซ์ซอร์สโค้ดสำหรับ Visual Basic 5.0 นั้นเป็นสิ่งสำคัญ เพราะถึงแม้ว่าในปัจจุบันโค้ดที่ถูกคอมไพล์แล้ว จะสามารถสร้างเป็นไฟล์ .EXE ในรูปแบบของโค้ดแท้ได้ก็ตาม แต่โปรแกรมที่เขียนด้วย Visual Basic 5.0 ก็ยังคงทำงานได้ช้ากว่าโปรแกรมที่เขียนด้วย Visual C++ อยู่พอสมควร ดังนั้นถ้าหากผู้อ่านรู้วิธีการทำออปติไมซ์ซอร์สโค้ดที่เหมาะสม ก็จะสามารถช่วยเพิ่มความเร็วของโปรแกรมได้มากทีเดียว

ส่วนโปรแกรมตัวอย่างในการทดสอบผลลัพธ์ของการออปติไมซ์ซอร์สโค้ดนั้น ผู้เขียนจะใช้ฟังก์ชันด้านเวลา (timer) จากไฟล์ไลบรารี Benmrk32.dll ซึ่งเป็นไฟล์ไดนามิกลิงก์ไลบรารีที่แท้จริง ในการจับเวลาในการประมวลผลโค้ดตัวอย่าง ซึ่งโดยปกติจะมีเวลาในการปฏิบัติการในระดับของมิลลิวินาที (หรือ 1/1000 วินาที) ซึ่งผู้เขียนจะได้ทำการอธิบายถึงฟังก์ชันทั้งหมดและวิธีการใช้งานอย่างละเอียดในตอนต่อไปครับ และขออนุญาตบอกกล่าวถึงผู้อ่านที่สนใจจะศึกษา Visual Basic 5.0 ด้วยตนเอง หรือต้องการทราบวิธีการใช้งานคอนโทรล ActiveX อย่างละเอียด ก็สามารถหาอ่านได้จากหนังสือ การใช้โปรแกรม Visual Basic 5.0 Professional ซึ่งคาดว่าจะสามารถจะออกจำหน่ายได้อย่างแน่นอนในราวปลาย ๆ เดือนกันยายน 2541 นี้ครับ โดยสามารถหาซื้อได้จากซีเอ็ดบุ๊คเซ็นเตอร์ หรือร้านหนังสือชั้นนำทั่วไป ในราคามิตรภาพในยุค IMF ครองเมืองครับ

การออปติไมซ์

ในปัจจุบันถึงแม้ว่า PC จะได้รับการพัฒนาอย่างรวดเร็วจนมาถึงยุคของ Pentium II ก็ตาม แต่ระดับการพัฒนาซอฟต์แวร์ก็ยังคงมีความล้าหลังตามการพัฒนาฮาร์ดแวร์อยู่ดี ซอฟต์แวร์ที่ทำงานโดยทั่วไปมีส่วนอย่างมากที่จะทำให้ผู้ใช้ไม่สามารถใช้งานฮาร์ดแวร์ได้อย่างเต็มที่ ทั้งนี้อาจเนื่องมาจากการที่ซอฟต์แวร์บางตัวใช้พลังการประมวลผลของ CPU ที่มากเกินความจำเป็น ซึ่งส่วนหนึ่งของสาเหตุดังกล่าวก็มาจากการที่ซอฟต์แวร์ไม่ได้รับการออปติไมซ์ซอร์สโค้ดนั่นเอง ดังนั้นเพื่อให้ซอฟต์แวร์ที่ผู้อ่านสร้างขึ้นมาไม่เป็นอุปสรรคในการแบ่งการใช้งาน CPU ร่วมกับซอฟต์แวร์ตัวอื่น ๆ ผู้อ่านก็มีความจำเป็นต้องทำการออปติไมซ์ซอร์สโค้ดให้สมบูรณ์ก่อนที่จะทำการคอมไพล์เพื่อสร้างเป็นไฟล์ .EXE และโดยเฉพาะซอฟต์แวร์ที่ได้รับการพัฒนาโดย Visual Basic เพราะจากการทดสอบของผู้เขียนพบว่า แต่ละแอพพลิเคชันที่พัฒนาโดย Visual Basic นั้นต้องการหน่วยความจำในการทำงานประมาณ 5 เมกะไบต์ ดังนั้นถ้าหากเครื่องคอมพิวเตอร์ของผู้อ่านติดตั้งหน่วยความจำอย่างเต็มที่ 128 เมกะไบต์ ผู้อ่านจะพบว่าจำนวนของการติดต่ออย่างทันทีทันใด จะสามารถกระทำได้เพียงประมาณ 20-25 จำนวนของการติดต่ออย่างทันทีทันใดเท่านั้น ดังนั้นถ้าหากซอฟต์แวร์ที่สร้างขึ้นมาได้รับการออปติไมซ์ซอร์สโค้ดอย่างเหมาะสม ก็จะช่วยชดเชยจุดด้อยในด้านดังกล่าวของ Visual Basic ได้มากทีเดียว

เราพูดถึงการออปติไมซ์มาพอสมควร คราวนี้ผู้เขียนก็จะขออธิบายว่า การออปติไมซ์หมายความว่าอย่างไร ในการเขียนซอฟต์แวร์ การออปติไมซ์หมายถึง วิธีหรือเทคนิคในการเขียนโค้ดเพื่อให้ซอฟต์แวร์ที่ได้สามารถทำงานได้อย่างรวดเร็ว (speed optimization) และ/หรือ สามารถทำงานได้โดยใช้ทรัพยากรของระบบน้อยที่สุด (resource optimization) ซึ่งจุดประสงค์ทั้งสองอย่างในบางครั้งก็สามารถที่จะทำได้พร้อม ๆ กัน แต่ในบางครั้งก็ไม่สามารถทำได้พร้อม ๆ กัน ดังนั้นโปรแกรมเมอร์ก็จำเป็นต้องมีการประเมินผลดีและผลเสียของผลลัพธ์ที่ได้ เพื่อดูความเหมาะสมสำหรับแต่ละสถานการณ์หรือข้อจำกัดต่าง ๆ ในการนำซอฟต์แวร์ไปใช้งาน ซึ่งในทางปฏิบัติการออปติไมซ์นั้นถือได้ว่าเป็นทั้งศาสตร์และศิลป์ ถึงแม้ว่าโดยหลักการจะมีเทคนิคการออปติไมซ์ที่คล้าย ๆ กัน แต่ในความเป็นจริงโปรแกรมเมอร์แต่ละท่านจะมีวิธีการประสมประสานเพื่อออปติไมซ์โค้ดที่ไม่เหมือนกันอย่างแน่นอน

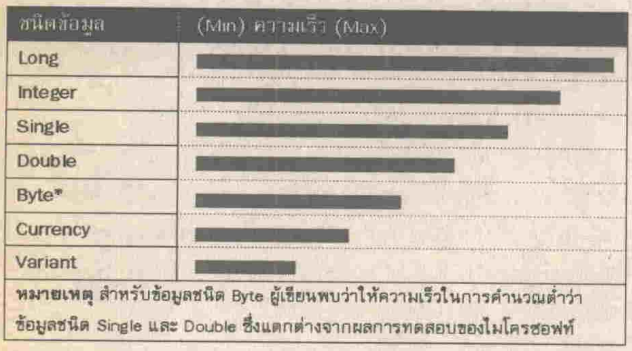
สำหรับภาพในมุมกว้างของการออปติไมซ์สำหรับ Visual Basic นั้น ผู้เขียนจะขอแบ่งออกเป็นกลุ่มใหญ่ ๆ 5 กลุ่ม ตามประสบการณ์ในการเขียนโปรแกรมในระดับ 32 บิต ดังต่อไปนี้

- การออปติไมซ์ในด้านความเร็วของการทำงานของซอฟต์แวร์ หรือที่เรียกกันว่า real speed optimization ซึ่งถือว่าเป็นหัวใจหลักที่สำคัญที่สุดในการออปติไมซ์
- การออปติไมซ์ในด้านขนาดของซอฟต์แวร์ (run-time size optimization) ซึ่งเหมาะสำหรับซอฟต์แวร์ที่พัฒนาโดย Visual Basic เป็นอย่างมาก ตามเหตุผลที่กล่าวมาแล้วข้างต้น
- การออปติไมซ์ในด้านปริมาณการอ้างอิงทรัพยากรระบบ (resource allocation optimization) ซึ่งเหมาะสำหรับการสร้างซอฟต์แวร์ที่ทำงานภายใต้เครื่องคอมพิวเตอร์ที่มีข้อจำกัดด้านหน่วยความจำ ซึ่งมักจะไม่ค่อยพบเห็นในปัจจุบันในสถานการณ์ที่แรมมีราคาถูกและตกต่ำลงเรื่อย ๆ
- การออปติไมซ์ในด้านความเร็วของการแสดงผล (display speed optimization) ซึ่งเป็นการออปติไมซ์ที่สำคัญมากสำหรับการสร้างซอฟต์แวร์ประเภทเกมหรือเพื่อความบันเทิง
- การออปติไมซ์ในด้านกราฟิก (graphic optimization) ซึ่งรวมทั้งการออปติไมซ์ขนาดหน่วยความจำหรือขนาดของกราฟิก และความเร็วในการแสดงผลกราฟิก ซึ่งในกรณีหลังก็จะเกี่ยวพันถึงความเร็วในการแสดงตัวของซอฟต์แวร์ในขณะโหลดโมดุลต่าง ๆ หรือเริ่มต้นทำงาน

ในโลกของความเป็นจริงผู้เขียนมั่นใจว่า ไม่มีโปรแกรมเมอร์คนใดที่จะสามารถทำการออปติไมซ์ซอฟต์แวร์ได้ครบทั้ง 5 กลุ่ม ในเวลาเดียวกัน เพราะการออปติไมซ์บางกลุ่มโปรแกรมเมอร์ก็มีความจำเป็นต้องผ่อนคลายโปรแกรมเมอร์บางกลุ่ม เช่น ถ้าหากผู้อ่านต้องการการออปติไมซ์ในด้านความเร็วของการทำงานของซอฟต์แวร์ โดยปกติซอฟต์แวร์ที่ถูกสร้างก็จะมีขนาดที่ใหญ่กว่าซอฟต์แวร์โดยทั่วไป เพราะการออปติไมซ์ในด้านความเร็วของการทำงานของซอฟต์แวร์ ผู้อ่านจำเป็นต้องเขียนโค้ดที่มีการกระโดดไปมาที่น้อยที่สุด และในที่สุดเราก็จะไม่สามารถหลีกเลี่ยงการเขียนโค้ดที่ซ้ำซ้อนได้เลย

ดังนั้นก่อนที่ผู้อ่านจะเริ่มต้นการออปติไมซ์ซอร์สโค้ด ผู้อ่านต้องมีการกำหนดเป้าหมายสำหรับซอฟต์แวร์เสียก่อนว่า ซอฟต์แวร์ที่ถูกสร้างขึ้นมาจะสามารถสร้างความพอใจให้กับผู้ใช้ได้อย่างไร ซึ่งเป็นเป้าหมายหลักในการสร้างซอฟต์แวร์โดยทั่วไป เช่น สมมติว่าผู้อ่านทำการพัฒนาซอฟต์แวร์ประเภทเกม สิ่งที่สำคัญที่สุดที่ต้องคำนึงก็คือ ความเร็วในการทำงานและแสดงผลกราฟิกของซอฟต์แวร์ และสิ่งที่มีความสำคัญน้อยที่สุดก็คือ การออปติไมซ์ในด้านปริมาณการอ้างอิงทรัพยากรระบบ เพราะโดยทั่วไปผู้ใช้ที่กำลังเล่นเกมมักจะไม่ใช้งานซอฟต์แวร์อื่น ๆ ไปพร้อม ๆ กัน เพราะซอฟต์แวร์เกมต้องการการปฏิสัมพันธ์ (interactive) กับผู้ใช้ในรูปแบบของ real-time นั่นเอง ดังนั้นเมื่อถึงตรงนี้ผู้เขียนก็จะขอสรุปว่าการออปติไมซ์ที่ดี ผู้อ่านต้องมีความเข้าใจถึงปัญหาที่แท้จริงที่คาดว่าจะเกิดขึ้น เพื่อที่จะได้ทำการออปติไมซ์ได้อย่างถูกต้อง

สำหรับ Visual Basic ที่มาพร้อมกับคอนโทรล ActiveX อย่างมากมายนั้น ในบางกรณีการออปติไมซ์ซอร์สโค้ดในด้านของความเร็วก็ไม่ใช่สิ่งที่จำเป็นต้องทำเพราะคุณสมบัติบางประการมีอยู่แล้วนั่นเอง เช่น สมมติว่าผู้อ่านต้องการออปติไมซ์เทคนิคการเรียงลำดับข้อมูล (sorting) ซึ่งในกรณีนี้ผู้อ่านสามารถใช้ความสามารถจากคุณสมบัติ



รูปที่ 1 แสดงความเร็วในการคำนวณข้อมูลชนิดตัวเลขของ Visual Basic

ชนิดข้อมูล	เวลา (วินาที)
Long	.051 (19.32%)
Integer	.069 (26.14%)
Single	.075 (28.04%)
Double	.114 (43.18%)
Byte	.125 (47.35%)
Currency	.135 (51.14%)
Variant	.264 (100.00%)

```
Dim a As Long
Dim o As Long, p As Integer

For a = -32767 To 32766
 o = 2
 o = o + 2
 o = o - 2
 o = o * 2
 o = o / 2
Next

For a = -32767 To 32766
 p = 2
 p = p + 2
 p = p - 2
 p = p * 2
 p = p / 2
Next
```

รูปที่ 2 แสดงผลเปรียบเทียบความเร็วในการคำนวณข้อมูลชนิดตัวเลขของ Visual Basic

Sorted ของคอนโทรล ListBox ในการเรียงลำดับข้อมูล จากนั้นจึงค่อยทำการอ่านข้อมูลที่ถูกเรียงลำดับเรียบร้อยแล้วกลับคืนในภายหลัง ดังนั้นถ้าหากผู้อ่านได้ศึกษาถึงคุณสมบัติทั้งหมดที่มากับ Visual Basic อย่างละเอียด ก็จะช่วยลดภาระการออปติไมซ์ในด้านต่าง ๆ ได้มากทีเดียว โดยเฉพาะคุณสมบัติที่มาพร้อมกับคอนโทรล ActiveX

การออปติไมซ์ในด้านความเร็วของการทำงานของซอฟต์แวร์

ด้านความเร็วของการทำงานของซอฟต์แวร์ นอกจากจะขึ้นกับคอมไพเลอร์แล้วยังขึ้นกับระบบปฏิบัติการอีกด้วย ดังนั้นการออปติไมซ์ในด้านความเร็วจึงต้องคำนึงถึงปัจจัยหลาย ๆ ด้าน สำหรับ Visual Basic ผู้เขียนก็จะขอเสนอจุดหลัก ๆ ที่สามารถออปติไมซ์ในด้านความเร็วของโค้ดสำหรับ Visual Basic อย่างได้ผล เพราะเป็นโค้ดที่พบบ่อยครั้งที่สุดในการเขียนโปรแกรม ดังต่อไปนี้

ใช้ตัวแปรชนิดเลขจำนวนเต็ม Long แทน Integer ในการคำนวณทุกประเภท

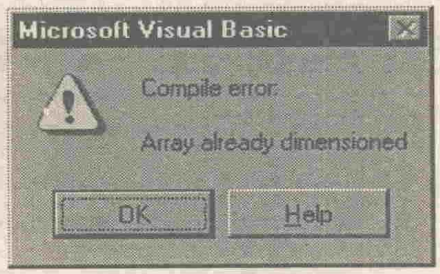
ในตอนแรก ๆ ที่ผู้เขียนเริ่มเปลี่ยนจากการเขียนโค้ดระดับ 16 บิต มาเป็นโค้ดระดับ 32 บิต ใหม่ ๆ ก็เคยเข้าใจผิดว่า ตัวแปรชนิดเลขจำนวนเต็ม Integer จะเป็นตัวแปรที่สามารถใช้ในการคำนวณได้รวดเร็วที่สุด แต่เมื่อผู้เขียนได้ทดลองทำการออปติไมซ์โค้ด กับพบว่าตัวแปรชนิดเลขจำนวนเต็ม Integer กลับทำงานได้ช้ากว่าชนิด Long ทั้งนี้เนื่องจากภายใต้ระบบปฏิบัติการวินโดวส์ 32 บิต (95, 98 หรือ NT) การอ้างอิงข้อมูลขนาด 4 ไบต์ (32 บิต) จะเป็นวิธีการอ้างอิงข้อมูลที่รวดเร็วที่สุด ด้วยเหตุนี้ตัวแปรชนิดเลขจำนวนเต็ม Long จึงสามารถถูกเข้าถึงได้เร็วกว่า Integer และจากการทดสอบข้อมูลชนิดตัวเลขทั้งหมดของ Visual Basic ก็พบว่าข้อมูลชนิดตัวเลขที่สามารถถูกคำนวณได้รวดเร็วที่สุดมีลำดับดังต่อไปนี้

ผลการทดสอบต่อไปนี้ เป็นการทดสอบโค้ดที่ถูกคอมไพล์เป็น .EXE แบบโค้ดไบนารีแล้ว โดยที่ตัวเลขเปอร์เซ็นต์เป็นการเปรียบเทียบกับเวลาที่ใช้ในการคำนวณข้อมูลชนิด Variant (100%) และโค้ดที่ใช้ทดสอบการคำนวณมีดังรูปที่ 2

จากการทดสอบในบางกรณีกลับพบว่าข้อมูลชนิดตัวเลข Single สามารถใช้ในการคำนวณได้เร็วกว่าข้อมูลชนิดตัวเลข Long เสียอีก ซึ่งผู้อ่านจะได้พบในโปรแกรมทดสอบที่จะกล่าวถึงในตอนต่อไป

หลีกเลี่ยงการใช้ตัวแปรชนิด Variant ให้มากที่สุด

เนื่องจากตัวแปรชนิด Variant สามารถใช้จัดเก็บได้ทั้งสตริงและตัวเลข ซึ่งเป็นข้อมูลชนิดพิเศษของ Visual Basic และได้ถูกกำหนดให้เป็นชนิดข้อมูล default ของ Visual Basic ดังนั้นถ้าหากผู้อ่านประกาศตัวแปรโดยไม่กำหนดชนิดข้อมูล Visual Basic ก็จะกำหนดให้เป็นข้อมูลชนิด Variant โดยอัตโนมัติทันที และจากตัวอย่างข้างต้นจะเห็นว่าเมื่อเปรียบเทียบระหว่างข้อมูลชนิดตัวเลขกับ Variant จะพบว่าตัวแปรข้อมูลชนิดตัวเลขจะสามารถใช้ในการคำนวณได้เร็วกว่าอย่างน้อยประมาณ 2 เท่า และถ้าหากเปรียบเทียบระหว่างตัวแปรข้อมูลชนิดสตริง String และ Variant ก็จะให้ผลเช่นเดียวกัน โดยที่ตัวแปรข้อมูลชนิดสตริงจะสามารถทำการประมวลผลได้รวดเร็วกว่า



รูปที่ 3 แสดงความเร็วในการประมวลผลข้อมูลชนิดสตริงของ Visual Basic

ดังนั้นเพื่อให้มั่นใจว่าตัวแปรที่ถูกประกาศอาจจะโดยประโยค Dim, ReDim, Private, Public หรือ Global ก็ตาม ไม่มีตัวแปรที่ถูกกำหนดให้เป็นข้อมูลชนิด Variant โดยอัตโนมัติ ผู้อ่านก็ควรที่จะใช้ประโยค Option Explicit ในบรรทัดแรกสุดของโมดูล (ทั้งฟอร์ม .frm และโมดูลมาตรฐาน .bas) ทุกครั้ง หรืออาจจะโดยการบังคับให้ Visual Basic เพิ่มโค้ดดังกล่าวให้โดยอัตโนมัติก็ได้เช่นกัน โดยการเลือกตัวเลข Require Variable Declaration ในแถบป้าย Editor ของไดอะล็อกบ็อกซ์ Options เท่านั้น

ใช้ค่าคงที่แทนตัวแปร

สำหรับข้อนี้โปรแกรมเมอร์ทั่วไปคงจะใช้กันบ่อย โดยเฉพาะเมื่อต้องการจัดเก็บค่าตัวเลข แต่การจัดเก็บค่าสตริงกลับไม่ค่อยพบว่ามีการใช้ค่าคงที่กันมากนัก ซึ่งการจัดเก็บค่าตัวเลขหรือค่าสตริงในรูปแบบของค่าคงที่ (โดยใช้ประโยค Const) นั้นนอกจากจะทำให้โปรแกรมทำงานได้รวดเร็วขึ้นแล้ว ยังเป็นการง่ายในการแก้ไขหรือตรวจสอบโค้ดอีกด้วย เพราะการแก้ไขใด ๆ ผู้อ่านสามารถที่จะทำการแก้ไขเฉพาะค่าที่กำหนดให้กับค่าคงที่ในประโยค Const เท่านั้น ส่วนสาเหตุที่ค่าคงที่สามารถถูกประมวลผลได้เร็วกกว่า ก็เนื่องจากค่าคงที่จะถูกคอมไพเลอร์แปลและให้ตำแหน่งที่แน่นอน ส่วนตัวแปรจะไม่ถูกกำหนดตำแหน่งที่แน่นอนในขณะคอมไพล์ แต่จะถูกคำนวณอีกครั้งในขณะปฏิบัติการ ด้วยเหตุนี้โปรแกรมจึงสามารถเข้าถึงค่าที่ถูกจัดเก็บแบบค่าคงที่ได้เร็วกว่าตัวแปร

ส่งผ่านตัวแปรไปยังโพรซีเดอร์แบบส่งค่า (ByVal) แทนการส่งแอดเดรส (ByRef)

สำหรับเทคนิคนี้มักจะไม่ค่อยมีใครทราบ เพราะประโยค ByVal และ ByRef มักจะไม่ถูกใช้ในการเขียนโปรแกรม ยกเว้นสำหรับผู้อ่านที่เคยเขียนโปรแกรมเพื่อลิงก์กับฟังก์ชันในไฟล์ .DLL เท่านั้น ก่อนอื่นผู้เขียนก็จะขออธิบายถึงข้อแตกต่างระหว่างประโยคทั้งสองเสียก่อน

ประโยค ByVal ใช้นำหน้าชื่อตัวแปรในโพรซีเดอร์ (Sub หรือ Function) เพื่อกำหนดให้ Visual Basic ส่งผ่านตัวแปรไปยังโพรซีเดอร์

ชนิดข้อมูล	(Min) ความเร็ว (Max)
Constant	
Variable	

รูปที่ 5 แสดงความเร็วในการประมวลผลค่าคงที่และตัวแปร

ชนิดข้อมูล	เวลา (วินาที)
Dim p As String, q As Variant For a = 1 To 10240 p = p & "1" Next For a = 1 To 10240 q = q & "1" Next	
String	4.714 (97.22%)
Variant	4.849 (100.00%)

รูปที่ 4 แสดงผลเปรียบเทียบความเร็วในการประมวลผลข้อมูลชนิดสตริงของ Visual Basic

ชนิดข้อมูล	เวลา (วินาที)
Const C1 = "SE-ED Co.,Ltd." Dim pf, qf For a = 1 To 10240 pf = C1 pf = "" Next For a = 1 To 10240 pf = qf pf = "" Next	
Constant	.286 (94.39%)
Variable	.303 (100.00%)

รูปที่ 6 แสดงผลเปรียบเทียบความเร็วในการประมวลค่าคงที่และตัวแปร

แบบส่งค่า ซึ่งจะส่งผลให้ค่าของตัวแปรที่ถูกแก้ไขภายในโพรซีเดอร์จะไม่มีผลต่อค่าของตัวแปรที่ถูกส่งให้กับโพรซีเดอร์ ในทางกลับกันประโยค ByRef ใช้นำหน้าชื่อตัวแปรในโพรซีเดอร์ (Sub หรือ Function) เพื่อกำหนดให้ Visual Basic ส่งผ่านตัวแปรไปยังโพรซีเดอร์แบบแอดเดรส ซึ่งเป็นวิธีปกติของ Visual Basic ซึ่งจะส่งผลให้ค่าของตัวแปรที่ถูกแก้ไขภายในโพรซีเดอร์จะมีผลต่อค่าของตัวแปรที่ถูกส่งให้กับโพรซีเดอร์

```
ส่งผ่านตัวแปรไปยังโพรซีเดอร์แบบส่งค่าโดยประโยค ByVal
a& = 9
AddValue a&, 12

' a& = 9
Debug.Print a&

Sub AddValue(ByVal parm1&, ByVal parm2&)
    parm1& = parm1& + parm2&
End Sub
```

การส่งค่า	(Min) ความเร็ว (Max)
ByVal	
ByRef	

รูปที่ 7 แสดงความเร็วในการประมวลผลการส่งผ่านตัวแปรไปยังโพรซีเดอร์แบบส่งค่าและส่งแอดเดรส

```
ส่งผ่านตัวแปรไปยังโพรซีเดอร์แบบส่งแอดเดรสโดยประโยค ByRef

a& = 9
AddValue a&, 12

' a& = 21
Debug.Print a&

        Sub AddValue(ByRef parm1&, ByRef parm2&)
            parm1& = parm1& + parm2&
        End Sub
```

การส่งค่า	เวลา (วินาที)
ByVal	.070 (97.22%)
ByVRef	.072 (100.00%)

```
For a = -32767 To 32766
  p = 0
  TestByVal p, q
Next

For a = -32767 To 32766
  p = 0
  TestByRef p, q
Next

Sub TestByVal(ByVal P1 As Long, ByVal P2 As Long)
  P1 = P1 + P2
End Sub

Sub TestByRef(ByRef P1 As Long, ByRef P2 As Long)
  P1 = P1 + P2
End Sub
```

รูปที่ 8 แสดงผลเปรียบเทียบความเร็วในการประมวลการส่งผ่านตัวแปรไปยังโพรซีเดอร์แบบส่งค่าและส่งแอดเดรส

ใช้ตัวแปรแทนคุณสมบัติของคอนโทรล ActiveX ในการวนลูป

ในกรณีที่ผู้อ่านจำเป็นต้องเขียนโค้ดเพื่อเข้าถึงหรือแก้ไขค่าคุณสมบัติของคอนโทรล ActiveX ต่าง ๆ บ่อยครั้งหรือมีการกระทำดังกล่าวภายในวงลูป เช่น For...Next เป็นต้น ซึ่งในกรณีนี้ผู้เขียนขอแนะนำให้ผู้อ่านทำการจัดเก็บค่าคุณสมบัติของคอนโทรล ActiveX ลงในตัวแปรใด ๆ นอกวงลูปเสียก่อน จากนั้นภายในวงลูปก็ให้ใช้การเข้าถึงหรือแก้ไขตัวแปรดังกล่าวแทน จนสิ้นสุดการวนวงลูป หลังจากที่ออกจากวงลูปแล้วจึงค่อยกำหนดค่าจากตัวแปรคืนให้กลับคุณสมบัติของคอนโทรล ActiveX ข้างต้นอีกครั้ง เทคนิคนี้จะมีส่วนช่วยในการเพิ่มความเร็วของโปรแกรมมากทีเดียว โดยเฉพาะสำหรับคอนโทรลที่ต้องมีการอัพเดตการแสดงผลทุกครั้งที่คุณสมบัติของคอนโทรล ActiveX มีการเปลี่ยนแปลง หรือคุณสมบัติของคอนโทรล ActiveX ที่กำลังแก้ไขเป็นคุณสมบัติที่เกี่ยวกับการแสดงผล เช่น คุณสมบัติ Text ของคอนโทรล TextBox เป็นต้น เพราะการลดจำนวนครั้งในการแก้ไขคุณสมบัติเหล่านี้จะช่วยลดการแสดงผลคอนโทรลของวินโดวส์นั่นเอง

โค้ดที่ใช้ในการทดสอบข้างล่าง เป็นการแสดงผลของการแก้ไขค่าคุณสมบัติ Text ของคอนโทรล TextBox โดยตรงกลับการแก้ไขโดยทางอ้อมโดยผ่านทางตัวแปรสตริง และมีข้อสังเกตประการหนึ่ง โดยก่อนที่จะทดสอบก็ขอให้ผู้อ่านทดลองลบทิ้งโค้ดบรรทัด Text1.Visible = False ก็จะพบว่าการซ่อนคอนโทรลก่อนการแก้ไขค่าคุณสมบัติ Text จะทำให้โปรแกรมทำงานเร็วกว่าการแก้ไขค่าคุณสมบัติ Text ในขณะที่คอนโทรล TextBox ยังแสดงผล ดังนั้นจึงสรุปได้ว่า การเข้าถึงหรือแก้ไขค่าคุณสมบัติของคอนโทรล ActiveX นั้น จะทำให้โปรแกรมทำงานช้าลงด้วยสาเหตุ 2 ประการ คือ การเข้าถึงหรือแก้ไขค่าคุณสมบัติสามารถกระทำได้ช้า และวินโดวส์ต้องเสียเวลาในการอัพเดตคอนโทรลทุกครั้งที่ค่าของคุณสมบัติที่เกี่ยวข้องเปลี่ยนแปลง

ข้อมูลชนิดตัวเลข	(Min) ความเร็ว (Max)
Variable	
Control Property	

รูปที่ 9 แสดงความเร็วในการประมวลผลตัวแปรและคุณสมบัติของคอนโทรล ActiveX

สำหรับเทคนิคนี้นอกจากจะสามารถประยุกต์ใช้กับคุณสมบัติของคอนโทรลแล้ว ผู้อ่านยังสามารถนำไปประยุกต์ใช้กับการเข้าถึงฟังก์ชัน (หรือคำสั่ง) ต่าง ๆ ของ Visual Basic ได้อีกด้วย โดยถ้าหากจำเป็นต้องเข้าถึงฟังก์ชันต่าง ๆ ของ Visual Basic บ่อย ๆ ก็ให้จัดเก็บค่าลงในตัวแปรเสียก่อน แล้วจึงค่อยเข้าถึงตัวแปร เพื่อลดเวลาที่ต้องสูญเสียในการเข้าถึงฟังก์ชันเหล่านั้นในไฟล์ไลบรารีของ Visual Basic (ไฟล์ Msvbvm50.dll)

ข้อมูลชนิดตัวเลข	เวลา (วินาที)
Text1.Visible = False pf = "" For a = 0 To 255 pf = pf & a Next Text1.Text = pf Text1.Text = "" For a = 0 To 255 Text1.Text = Text1.Text & a Next Text1.Visible = True	
Variable	.050 (06.98%)
Control Property	.716 (100.00%)

รูปที่ 10 แสดงผลเปรียบเทียบความเร็วในการประมวลตัวแปรและคุณสมบัติของคอนโทรล ActiveX

การออปติไมซ์ในด้านขนาดของซอฟต์แวร์

การลดขนาดของซอฟต์แวร์ก็นับว่าเป็นสิ่งที่มีความสำคัญพอสมควร โดยเฉพาะสำหรับ Visual Basic ที่ต้องการไฟล์ไลบรารี Msvbvm50.dll ที่มีขนาดใหญ่ในการทำงาน และเพื่อลดความเทอะทะของซอฟต์แวร์ที่สร้างโดย Visual Basic ไมโครซอฟท์ถึงกับมีการกำหนดให้เวอร์ชันต่อ ๆ ไปของ Visual Basic ไม่จำเป็นต้องใช้ไฟล์ไลบรารี Msvbvm50.dll ในการสร้างชุด Setup เพราะไมโครซอฟท์ได้ใส่ไฟล์ดังกล่าวลงในวินโดวส์อยู่แล้ว ซึ่งผู้อ่านคงพอที่จะคาดการณ์ได้ว่าไมโครซอฟต์ได้กำหนดให้ Visual Basic มีความสำคัญอย่างไรในอนาคต

ลดการใช้โค้ดที่ซ้ำซ้อน

ในการเขียนโค้ดเพื่อกระทำอย่างใดอย่างหนึ่งอย่างซ้ำซ้อน ผู้อ่านควรที่จะสร้างชุดโค้ดดังกล่าวลงในโพรซีเดอร์แล้วจึงเรียกใช้โพรซีเดอร์ดังกล่าว แทนที่จะเขียนชุดโค้ดดังกล่าวทุกครั้งที่ต้องการเรียกใช้ เพราะจะทำให้ขนาดของไฟล์ .EXE ที่คอมไพล์มีขนาดเล็กลงมากทีเดียว ทั้งนี้เนื่องจากการเขียนชุดโค้ดซ้ำซ้อน จะทำให้คอมไพเลอร์แปลโค้ดดังกล่าวแบบหนึ่งต่อหนึ่ง จึงทำให้ขนาดของไฟล์ .EXE มีขนาดโตขึ้นตามชุดโค้ดซ้ำซ้อน ในขณะเดียวกันถ้าหากผู้อ่านเลือกสร้างชุดโค้ดดังกล่าวลงในโพรซีเดอร์ และเรียกใช้โพรซีเดอร์แทน คอมไพเลอร์ก็จะแปลชุดโค้ดในโพรซีเดอร์เพียงครั้งเดียว ส่วนในการเรียกใช้โพรซีเดอร์แต่ละครั้ง คอมไพเลอร์จะไม่แทรกชุดโค้ดในโพรซีเดอร์ลงในตำแหน่งดังกล่าว แต่จะเป็นการเขียนโค้ดเพื่ออ้างอิงถึงแอดเดรสของโพรซีเดอร์ที่ถูกเรียกใช้เท่านั้น จึงทำให้โค้ดที่ได้มีขนาดเล็กลงมาก

แต่ในขณะเดียวกันการเรียกใช้โพรซีเดอร์บ่อยครั้ง จะทำให้ความเร็วในการทำงานของโปรแกรมลดลงตามไปด้วย ทั้งนี้เนื่องจากในการเรียกใช้โพรซีเดอร์แต่ละครั้ง ระบบจะต้องมีการจัดเก็บค่าต่าง ๆ ที่เกี่ยวข้องกับโพรซีเดอร์ลงในสแต็ก (stack) ซึ่งเป็นระบบหน่วยความจำที่มีการเข้าถึงได้ช้ามาก จึงทำให้การเรียกใช้โพรซีเดอร์แต่ละครั้งต้องเสียเวลาในการจัดเก็บและเรียก (push and pop) ค่าในสแต็ก

ลดปริมาณฟอร์มในโปรแกรม

ฟอร์มเป็นส่วนที่ Visual Basic ใช้ในการสื่อการทำงานกับผู้ใช้ ดังนั้น ในการสร้างซอฟต์แวร์จึงไม่อาจที่จะหลีกเลี่ยงการใช้ฟอร์ม แต่ผู้อ่านก็ควรที่จะใช้ฟอร์มในปริมาณเท่าที่จำเป็นเท่านั้น เพราะแต่ละฟอร์มที่เพิ่มเข้าไปในโปรแกรม ก็เท่ากับเป็นการเพิ่มขนาดของไฟล์ .EXE และหน่วยความจำในขณะทำงาน

สำหรับตัวผู้เขียนจะใช้เทคนิคการแชร์ฟอร์มประเภทที่ใช้ในการแสดงข้อความ หรือที่มีหน้าตาคล้าย ๆ กัน เช่น สมมติว่าในโปรแกรมเรามีฟอร์มสำหรับแสดงข้อมูลเกี่ยวกับโปรแกรม หรือที่ทั่วไปเรียกกันว่า About dialog box และฟอร์มสำหรับแสดงข้อผิดพลาดต่าง ๆ (Error Message dialog box) ผู้อ่านก็ไม่จำเป็นที่จะต้องสร้างฟอร์มทั้งสองประเภทลงในโปรแกรม แต่สามารถที่จะใช้ฟอร์มเพียงฟอร์มเดียว และใช้วิธีการเขียนโค้ดเพื่อแสดงผลเฉพาะส่วนที่เกี่ยวข้อง และซ่อนส่วนที่ไม่เกี่ยวข้องแทน โดยในรูปที่ 6 แสดงฟอร์มที่ใช้ร่วมกันระหว่างการใช้เป็น About dialog box และ Error Message dialog box โดยในการที่จะควบคุมว่าส่วนใดจะถูกแสดงผล ก็จะใช้คอนโทรล Frame เป็นตัวรวมชุดคอนโทรลที่ต้องการแสดงผล โดยในที่นี้ผู้เขียนใช้คอนโทรล Frame1 สำหรับแสดงผลเป็น About dialog box และคอนโทรล Frame2 สำหรับแสดงผล Error Message dialog box และใช้ค่าที่กำหนดให้กับคุณสมบัติ Tag ของฟอร์ม เป็นตัวบอกว่าในขณะนั้นต้องการที่จะใช้ฟอร์มในรูปแบบใด โดยก่อนที่ผู้อ่านจะทำการแสดงผลฟอร์มผู้อ่านก็ต้องกำหนดค่าให้กับคุณสมบัติ Tag ของฟอร์ม ก่อนเสมอ

```
โค้ดส่วนที่เรียกใช้ฟอร์ม About.frm
' Use as About dialog box.
frmAbout.Tag = "About"
frmAbout.Show
...
Unload frmAbout

โค้ดภายในฟอร์ม About.frm (frmAbout)
Private Sub Form_Activate()
 If Me.Tag = "About" Then
 Frame1.Visible = True
 Frame2.Visible = False
 Else
 Frame1.Visible = False
 Frame2.Visible = True
 End If
End Sub
```

แยกกราฟิกออกจากคอนโทรล PictureBox หรือคอนโทรล Image

กราฟิกที่ผู้อ่านกำหนดให้กับคอนโทรล PictureBox หรือคอนโทรล Image ในขณะที่กำลังเขียนโค้ด จะถูกกำหนดเป็นส่วนหนึ่งของไฟล์ .EXE ถ้าหากไฟล์ดังกล่าวถูกคอมไพล์ ดังนั้นถ้าหากกราฟิกที่ผู้อ่านกำหนดเป็นกราฟิกที่มีขนาดใหญ่หรือมีจำนวนสีมาก เช่น 24 บิต เป็นต้น ก็จะทำให้ไฟล์ .EXE มีขนาดโตขึ้นตามขนาดของกราฟิกที่ใส่ลงในคอนโทรลดังกล่าว แต่ผู้อ่านควรที่จะแยกกราฟิกออกเป็นไฟล์ต่างหากและค่อยเขียนโค้ดเพื่อโหลดกราฟิกให้กับคอนโทรลดังกล่าวในขณะกำลังทำงาน ซึ่งเทคนิคนี้นอกจากจะทำให้ไฟล์ .EXE มีขนาดเล็กลงแล้ว ยังทำให้เวลาที่ใช้ในการโหลดโปรแกรมมายังหน่วยความจำทั้งในขณะเริ่มต้นรันโปรแกรม หรือการโหลดฟอร์มที่บรรจุคอนโทรล PictureBox หรือคอนโทรล Image ดังกล่าวในภายหลัง ลดลงเป็นอย่างมาก ซึ่งผู้อ่านสามารถทดสอบได้โดยการกำหนดกราฟิกจากไฟล์ .bmp แบบ 24 บิต ที่มีขนาดโต ๆ ให้กับคอนโทรลดังกล่าวแล้วคอมไพล์เป็นไฟล์ .EXE จะเห็นว่าการโหลดโปรแกรมในขณะเริ่มต้นจะกินเวลานานมากทีเดียว

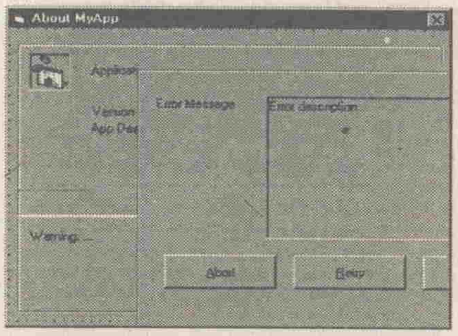
แต่การแยกกราฟิกออกจากคอนโทรล PictureBox หรือคอนโทรล Image จะทำให้โปรแกรมทำงานช้าลงเล็กน้อย ในขณะที่มีเรียกใช้โค้ดเพื่อโหลดกราฟิกให้กับคอนโทรล เพราะ Visual Basic ต้องทำการโหลดกราฟิกจากดิสก์แทนที่จะสามารถเข้าถึงในหน่วยความจำได้ทันทีนั่นเอง

ลดการมีโค้ดที่ไม่ถูกใช้งานในโปรแกรม

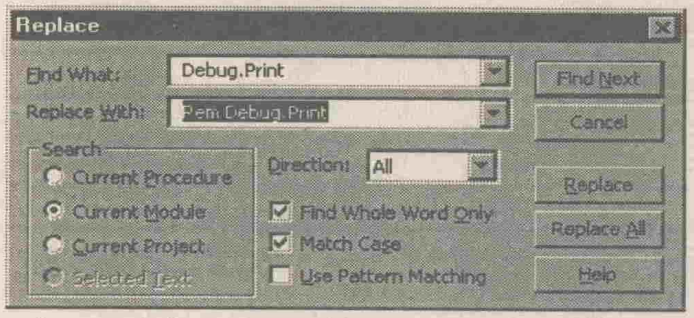
โค้ดที่ไม่ถูกใช้งาน (dead code) หรือโค้ดที่ไม่ถูกเรียกใช้เลยภายในโปรแกรม โดยปกติอาจเกิดเนื่องจากการเขียนโค้ดเพื่อการทดสอบการทำงานของโปรแกรม หรือตรวจสอบค่าตัวแปรต่าง ๆ ภายในโปรแกรม หรืออาจเกิดเนื่องจากโค้ดมีการแก้ไขหลาย ๆ ครั้ง และต้องการที่จะจัดเก็บโค้ดเก่าเอาไว้ (โดยปกติมักใส่ไว้ในส่วนการตรวจสอบเงื่อนไข เช่น Select Case... End Select) เป็นต้น โค้ดเหล่านี้จะทำให้โปรแกรมที่ได้มีขนาดโตขึ้นโดยไม่จำเป็น และยังทำให้ยากต่อการแก้ไขในภายหลังอีกด้วย

สำหรับในกรณีนี้ผู้เขียนจะขอเน้นพิเศษเฉพาะโค้ดที่เกี่ยวข้องกับออบเจ็กต์ Debug ซึ่งเป็นออบเจ็กต์ที่ใช้ในการดีบักหรือตรวจสอบค่าของตัวแปรต่าง ๆ ซึ่งโดยปกติโค้ดหรือตัวแปรทั้งหลายที่ตามหลังออบเจ็กต์ Debug จะถูก Visual Basic ประมวลผลเฉพาะเมื่อผู้อ่านรันโปรแกรมภายในเอนไวรอนเมนต์ของ Visual Basic เท่านั้น แต่เมื่อโปรแกรมถูกคอมไพล์เป็นไฟล์ .EXE โค้ดส่วนดังกล่าวจะไม่ถูกประมวลผลหรือแสดงผล จึงทำให้ผู้อ่านบางท่านคิดว่าโค้ดส่วนดังกล่าวจะไม่ถูกคอมไพล์ลงในไฟล์ .EXE เช่นกัน ซึ่งเป็นความเข้าใจที่ถูกต้องแต่ยังไม่ครบถ้วน เพราะในกรณีของการใช้ประโยค Debug.Print เพื่อแสดงผลฟังก์ชันต่าง ๆ นั้น โค้ดส่วนดังกล่าวจะยังคงถูกคอมไพล์ลงในไฟล์ .EXE และเมื่อทำงานโค้ดส่วนดังกล่าวก็จะยังคงถูกปฏิบัติตามปกติ เพียงแต่ค่าที่ได้จากฟังก์ชันจะไม่ถูกแสดงผลเท่านั้น ดังนั้นนอกจากจะทำให้โปรแกรมที่ได้มีขนาดโตขึ้นแล้ว ยังทำให้โปรแกรมทำงานได้ทำงานช้าลงอีกด้วย แต่สำหรับในกรณีที่ใช้ประโยค Debug.Print เพื่อแสดงผลค่าตัวแปร โค้ดส่วนดังกล่าวจะไม่ถูกคอมไพล์ลงในไฟล์ .EXE

ดังนั้นถ้าหากผู้อ่านมีการใช้ประโยค Debug.Print ภายในโปรแกรมก่อนที่จะคอมไพล์เป็นไฟล์ .EXE ผู้อ่านควรที่จะลบทิ้งหรือกำหนดประโยค Rem นำหน้าประโยค Debug.Print ทุกครั้ง สำหรับตัวผู้เขียนเห็นว่าการกำหนดประโยค Rem นำหน้าจะเป็นวิธีที่ง่ายที่สุด เพราะผู้อ่านสามารถใช้คำสั่ง Replace ของเอนไวรอนเมนต์ Visual Basic ช่วยในการค้นหาและเปลี่ยนแทนที่ประโยค Rem นำหน้าประโยค Debug.Print ได้โดยง่าย และในทางกลับกันยังสามารถใช้วิธีเดียวกันเพื่อลบทิ้งประโยค Rem ออกจากประโยค Debug.Print อีกด้วย



รูปที่ 11 แสดงความเร็วในการประมวลผลตัวแปรและคุณสมบัติของคอนโทรล ActiveX



รูปที่ 12 แสดงคำสั่ง Replace ของเอนไวรอนเมนต์ Visual Basic ช่วยในการค้นหาและเปลี่ยนแทนที่ประโยค Rem

การออปติไมซ์ในด้านปริมาณการอ้างอิงทรัพยากรระบบ

สำหรับคำว่าทรัพยากรระบบในที่นี้ผู้เขียนจะหมายถึงหน่วยความจำ ซึ่งเป็นส่วนสำคัญอย่างมากรองจาก CPU ที่เป็นตัวกำหนดความเร็วในการทำงานของวินโดวส์ เนื่องจากวินโดวส์เป็นระบบกราฟิกดังนั้นจึงต้อง

อาศัยหน่วยความจำเป็นอย่างมากในการทำงานหรือแสดงผล ดังนั้นถ้าหากผู้อ่านสามารถออกแบบโปรแกรมที่มีความต้องการทรัพยากรระบบในขณะทำงานน้อยที่สุด โดยปกตินอกจากจะทำให้ขนาดของไฟล์ .EXE เล็กลงแล้ว ยังทำให้โปรแกรมทำงานได้รวดเร็วอีกด้วย

ลดจำนวนฟอร์มที่ถูกโหลดมายังหน่วยความจำ

โดยปกติในแต่ละโปรแกรมจะประกอบด้วยฟอร์มมากกว่าหนึ่งฟอร์ม และในขณะที่โปรแกรมทำงานวินโดวส์จะยังคงไม่โหลดฟอร์มทั้งหมดของ Visual Basic มายังหน่วยความจำ นอกจากโค้ดที่มีการอ้างอิงถึงฟอร์มเหล่านั้นจะถูกปฏิบัติเท่านั้นวินโดวส์ถึงจะโหลดฟอร์มจากดิสก์ (จากบางส่วนของไฟล์ .EXE นั่นเอง) มายังหน่วยความจำ ดังเช่น โค้ดข้างล่างฟอร์ม frmAbout จะถูกโหลดมายังหน่วยความจำทันทีที่ประโยค Load frmAbout ถูกประมวลผล แต่เมื่อสิ้นสุดขั้นตอนนี้ฟอร์ม frmAbout ก็จะยังคงไม่ถูกแสดงผล จนกว่าโค้ดประโยค frmAbout.Show จะถูกประมวลผลเท่านั้น และในขณะเดียวกันโค้ดประโยค frmAbout.Hide จะทำให้ฟอร์ม frmAbout ถูกซ่อนแต่ยังไม่ถูกลบทิ้งออกจากหน่วยความจำ จนกว่าโค้ดประโยค Unload frmAbout จะถูกประมวลผลเท่านั้นฟอร์มจึงจะถูกลบทิ้งออกจากหน่วยความจำ

```
Private Sub Command1_Click()
 Load frmAbout
 frmAbout.Show
 …
 frmAbout.Hide
 Unload frmAbout
End Sub
```

ดังที่กล่าวมาแล้วฟอร์มจะถูกโหลดมายังหน่วยความจำเมื่อมีการเขียนโค้ดอ้างอิงถึงฟอร์ม ดังนั้นไม่เฉพาะโดยการใช้คำสั่ง Load เท่านั้นที่จะทำให้ฟอร์มถูกโหลดมายังหน่วยความจำ แต่ไม่ว่าผู้อ่านจะเขียนโค้ดอ้างอิงถึงคุณสมบัติ หรือวิธีใด ๆ ของฟอร์มก็ตามฟอร์มก็จะถูกโหลดมายังหน่วยความจำและแสดงผลทันที

ฟอร์มหนึ่ง ๆ นอกจากจะต้องการพื้นที่สำหรับโค้ดภายในฟอร์มในการประมวลผลแล้ว ยังต้องการทรัพยากรอื่น ๆ ของระบบในการแสดงผลกราฟิก ดังนั้นฟอร์มจึงเป็นส่วนหนึ่งขององค์ประกอบของโปรแกรมที่ใช้ทรัพยากรระบบมากที่สุด ดังนั้น

ในการเขียนโค้ดผู้เขียนจึงขอแนะนำให้ใช้ประโยค Unload เพื่อลบทิ้งฟอร์มที่ไม่จำเป็นที่จะต้องใช้งานในขณะนั้น ๆ ออกจากหน่วยความจำทุกครั้ง ซึ่งจะช่วยลดภาระการใช้ทรัพยากรระบบลงไปได้มากทีเดียว แต่ในทางตรงข้ามการเขียนโค้ดเพื่อลบทิ้งฟอร์มออกจากหน่วยความจำนั้น จะเป็นการเพิ่มงานให้โปรแกรมซึ่งจะส่งผลให้โปรแกรมทำงานช้าลงใน 2 ขั้นตอน

1. โปรแกรมต้องทำการคืนกลับ (disallocate) ทรัพยากรระบบที่ถูกอ้างอิงเอาไว้โดยฟอร์มดังกล่าวให้กับวินโดวส์ เมื่อฟอร์มถูกลบทิ้งออกจากหน่วยความจำ

2. โปรแกรมต้องทำการอ้างสิทธิ์ (allocate) ทรัพยากรระบบที่ต้องการสำหรับฟอร์มจากวินโดวส์ เมื่อต้องการโหลดฟอร์มจากดิสก์มายังหน่วยความจำ

ด้วยสาเหตุดังกล่าวจึงทำให้โปรแกรมทำงานช้าลงเมื่อมีการโหลดและลบทิ้งฟอร์มจากหน่วยความจำบ่อย ๆ ดังนั้นในทางปฏิบัติผู้เขียนขอแนะนำให้ผู้อ่านคำนึงถึงความจำเป็นและผลได้ผลเสียเป็นหลัก (trade-off) โดยถ้าหากฟอร์มมีขนาดพอประมาณและมีการเรียกใช้บ่อย ๆ ก็ไม่จำเป็นต้องใช้เทคนิคข้อนี้ แต่ถ้าหากฟอร์มมีขนาดใหญ่หรือมีการเรียกใช้ไม่บ่อยครั้ง และต้องไม่มีผลต่อการสื่อการใช้งานแบบปฏิสัมพันธ์กับผู้ใช้ทันที ผู้อ่านก็ควรที่จะกระทำตามเทคนิคข้อนี้

ใช้คอนโทรล Label แทนคอนโทรล TextBox

สำหรับเทคนิคในข้อนี้มีข้อจำกัดอยู่มาก เนื่องจากคอนโทรลทั้งสองมีวัตถุประสงค์การใช้งานที่แตกต่างกัน แต่จากประสบการณ์กลับพบว่ามีโปรแกรมเมอร์หลายท่านใช้คอนโทรลทั้งสองแทนกันโดยเฉพาะการใช้คอนโทรล TextBox เพื่อแสดงผลข้อความแทนคอนโทรล Label ทั้งนี้เนื่องจากสามารถที่จะเลื่อนพื้นที่แสดงผลได้นั่นเอง โดยอาจจะไม่ทราบว่าคอนโทรลทั้งสองใช้ทรัพยากรระบบที่แตกต่างกัน โดยคอนโทรล TextBox จะมีความต้องการทรัพยากรระบบมากกว่าคอนโทรล Label ดังนั้น ถ้าหากเป็นไปได้ถ้าหากผู้อ่านที่จะใช้คอนโทรล TextBox เพื่อวัตถุประสงค์ในการแสดงผลข้อความเท่านั้น ก็ควรที่จะใช้ คอนโทรล Labe. แทน เพราะนอกจากจะสามารถแสดงผลได้รวดเร็วกว่าแล้ว ยังมีความต้องการทรัพยากรระบบของวินโดวส์น้อยกว่าอีกด้วย

ใช้ตัวแปรอะเรย์แบบไดนามิกแทนแบบสแตติก

สำหรับวิธีนี้ค่อนข้างเป็นเทคนิคพอสมควร ซึ่งเป็นเทคนิคที่ผู้เขียนได้นำมาใช้ตั้งแต่เมื่อมีการสร้างโปรแกรมแก้ไขฟอนต์กราฟิกสำหรับ QuickBASIC (โปรแกรม BEQB.EXE จัดพิมพ์เป็นหนังสือโดยบริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน)) และยังคงเป็นเทคนิคที่สามารถนำมาใช้กับ Visual Basic ได้อย่างดีทีเดียว

สำหรับความหมายของคำว่าตัวแปรอะเรย์แบบไดนามิกแทนแบบสแตติก มีดังต่อไปนี้

ตัวแปรอะเรย์แบบไดนามิก

ตัวแปรอะเรย์แบบไดนามิก หมายถึง ตัวแปรอะเรย์ที่สามารถถูกแก้ไขขนาดจำนวนสมาชิกของอะเรย์ได้ในขณะที่โปรแกรมทำงาน โดยการเขียนโค้ดเพื่อทำการแก้ไข ดังนั้นตัวแปรอะเรย์แบบไดนามิกจึงมี

ความยืดหยุ่นสูงในการอ้างอิงหน่วยความจำสำหรับจัดเก็บข้อมูลของอะเรย์ เพราะถ้าหากผู้อ่านต้องการเพื่อจำนวนสมาชิกของอะเรย์ ผู้อ่านก็สามารถเขียนโค้ดเพื่อแก้ไขได้ทันที โดยที่ข้อมูลเดิมของสมาชิกที่มีอยู่แล้วจะไม่สูญหาย

ในทางปฏิบัติการสร้างตัวแปรอะเรย์แบบไดนามิก ผู้อ่านสามารถกระทำได้โดยการประกาศตัวแปรอะเรย์ด้วยประโยค Dim ในส่วนหลักการประกาศ (ส่วนของโค้ดที่อยู่นอกโพรซีเดอร์) โดยไม่ต้องระบุจำนวนสมาชิก เช่น โค้ดบรรทัด Dim Arr() As String เป็นการสร้างตัวแปรอะเรย์แบบไดนามิก Arr เป็นต้น จากนั้นผู้อ่านก็สามารถที่จะใช้ประโยค ReDim เพื่อที่จะปรับขนาดของตัวแปรอะเรย์โดยอาจจะปรับเพิ่มหรือปรับลดก็ได้ ซึ่งในการประกาศตัวแปรอะเรย์ด้วยประโยค ReDim นั้นผู้อ่านสามารถที่จะใช้ประโยค Preserve เพื่อกำหนดให้ข้อมูลเดิมของสมาชิกอะเรย์ยังคงอยู่ได้อีกด้วย

สำหรับโค้ดตัวอย่างข้างล่างได้มีการสร้างตัวแปรอะเรย์แบบไดนามิก Arr ในส่วนหลักการประกาศของฟอร์ม จากนั้นภายในโพรซีเดอร์เหตุการณ์ Command1_Click ได้มีการกำหนดให้ตัวแปรอะเรย์แบบไดนามิก Arr มีขนาด 2 สมาชิก และภายในโพรซีเดอร์เหตุการณ์ Command2_Click ได้มีการกำหนดให้ตัวแปรอะเรย์แบบไดนามิก Arr มีขนาด 4 สมาชิก โดยที่ข้อมูลของสมาชิกลำดับที่ 1 และ 2 จะไม่ถูกลบทิ้ง ดังนั้นเมื่อผู้อ่านทดสอบรันภายในเอนไวรอนเมนต์ของ Visual Basic ข้อมูลของแต่ละสมาชิกของตัวแปรอะเรย์แบบไดนามิก Arr ก็จะถูกแสดงออกทางหน้าต่างปฏิบัติทันที

```
Option Explicit
Dim Arr() As String

Private Sub Command1_Click()
 ReDim Arr(1 To 2) As String

 ' Arr(1) = "", Arr(1) = ""
 Debug.Print Arr(1), Arr(2)

 Arr(1) = "Data 1"
 Arr(2) = "Data 2"

 ' Arr(1) = "Data 1", Arr(1) = "Data 2"
 Debug.Print Arr(1), Arr(2)
End Sub

Private Sub Command2_Click()
 ReDim Preserve Arr(1 To 4) As String

 Arr(3) = "Data 3"
 Arr(4) = "Data 4"

 ' Arr(1) = "Data 1", Arr(1) = "Data 2"
 Debug.Print Arr(1), Arr(2)
 ' Arr(1) = "Data 3", Arr(1) = "Data 4"
 Debug.Print Arr(3), Arr(4)
End Sub
```

ตัวแปรอะเรย์แบบสแตติก

ตัวแปรอะเรย์แบบสแตติก หมายถึง ตัวแปรอะเรย์ที่ไม่สามารถแก้ไขขนาดจำนวนสมาชิกของอะเรย์ได้ในขณะที่โปรแกรมทำงาน โดยการเขียนโค้ดเพื่อทำการแก้ไข ดังนั้นตัวแปรอะเรย์แบบสแตติกจึงไม่มีความยืดหยุ่นสูงในการอ้างอิงหน่วยความจำสำหรับจัดเก็บข้อมูลของอะเรย์ เพราะการกำหนดจำนวนสมาชิกสามารถกระทำได้เพียงครั้งแรกเพียงครั้งเดียวเท่านั้น

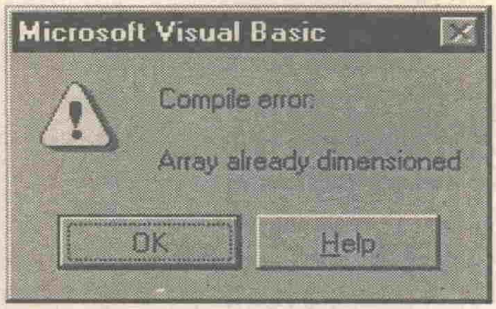
ในทางปฏิบัติการสร้างตัวแปรอะเรย์แบบสแตติก ผู้อ่านสามารถกระทำได้โดยการประกาศตัวแปรอะเรย์ด้วยประโยค Dim โดยต้องระบุจำนวนสมาชิกทันที เช่น โค้ดบรรทัด Dim Arr(1 To 10) As String เป็นการสร้างตัวแปรอะเรย์แบบสแตติก Arr เป็นต้น ดังนั้นในส่วนต่อไปของโปรแกรมผู้อ่านจะไม่สามารถใช้ประโยค ReDim เพื่อที่จะปรับขนาดของตัวแปรอะเรย์โดยการปรับเพิ่มหรือปรับลดอีกต่อไป

จากโค้ดตัวอย่างข้างล่างก็ขอให้ผู้อ่านทดลองรันภายในเอนไวรอนเมนต์ของ Visual Basic จะพบว่าเมื่อผู้อ่านคลิกที่ปุ่มคำสั่ง Command1 โพรซีเดอร์เหตุการณ์ Command1_Click ก็จะถูกเรียก และ Visual Basic จะแสดงข้อผิดพลาดดังในรูปที่ 8 ทันที ซึ่งเป็นข้อผิดพลาดที่โค้ดบรรทัด ReDim Arr(1 To 2) As String แต่ Visual Basic จะแสดงแถบสว่างที่บรรทัด Private Sub Command1_Click()

```
Option Explicit
Dim Arr(1 To 2) As String

Private Sub Command1_Click()
 ReDim Arr(1 To 2) As String

 ' Arr(1) = "", Arr(1) = ""
 Debug.Print Arr(1), Arr(2)

 Arr(1) = "Data 1"
 Arr(2) = "Data 2"

 ' Arr(1) = "Data 1", Arr(1) = "Data 2"
 Debug.Print Arr(1), Arr(2)
End Sub
```

กำหนดค่า Nothing ให้ตัวแปรออบเจ็กต์

ตัวแปรออบเจ็กต์เป็นตัวแปรที่ใช้ในการสร้าง (อ้างอิง) ออบเจ็กต์ของ Visual Basic เช่น การสร้างตัวอย่างสำเนา (instance) ฟอร์มหรือคอนโทรล ActiveX เป็นต้น ดังเช่นในโค้ดตัวอย่างข้างล่างเป็นการสร้างตัวแปรออบเจ็กต์ FrmObj เพื่อสร้างตัวอย่างสำเนาฟอร์ม Form1 ขึ้นมาใหม่ เป็นต้น ซึ่งผู้อ่านสามารถเขียนโค้ดเข้าถึงคุณสมบัติหรือวิธีต่าง ๆ ของตัวอย่างสำเนาฟอร์ม Form1 ที่ถูกสร้างขึ้นมาใหม่ โดยผ่านทางตัวแปรออบเจ็กต์ FrmObj อย่างเช่น การแสดงผลตัวอย่างสำเนาฟอร์ม Form1 โดยโค้ดดังนี้ FrmObj.Show



รูปที่ 13 แสดงข้อผิดพลาดที่รายงานโดยเอนไวรอนเมนต์ Visual Basic

โปรแกรมเมอร์โดยส่วนใหญ่มักคิดว่า ตัวแปรออบเจ็กต์จะถูกลบทิ้งทันทีที่ได้มีการเขียนโค้ดเพื่อทำลายหรือลบทิ้งตัวอย่างสำเนาฟอร์มหรือคอนโทรล ActiveX ออกจากหน่วยความจำ ดังเช่น โค้ดบรรทัด Unload FrmObj ในตัวอย่างข้างล่าง เป็นต้น แต่ความคิดดังกล่าวเป็นความเข้าใจที่ผิด เพราะถึงแม้ว่าตัวอย่างสำเนาฟอร์มหรือคอนโทรล ActiveX จะถูกลบทิ้งออกจากหน่วยความจำแล้วก็ตาม แต่ในความเป็นจริง Visual Basic จะยังคงรักษาสถานะของตัวแปรออบเจ็กต์ดังกล่าวเอาไว้ จนกว่าโปรแกรมจะสิ้นสุดการทำงาน ดังนั้นหน่วยความจำที่ถูกตัวแปรออบเจ็กต์อ้างอิงเอาไว้ก็จะยังคงไม่ถูกปล่อยคืนให้กับวินโดวส์ ดังนั้นในการยกเลิกการอ้างอิงหน่วยความจำที่ถูกอ้างอิงโดยตัวแปรออบเจ็กต์ ผู้อ่านจะต้องกำหนดค่า Nothing ให้กับตัวแปรออบเจ็กต์ ดังตัวอย่างของโค้ดในโพรซีเดอร์เหตุการณ์ Command2_Click ต่อไปนี้

```
Option Explicit
' Create object variable (reference to Form1)
Dim FrmObj As New Form1

Private Sub Command1_Click()
 FrmObj.Caption = "New Form1 object"
 FrmObj.Show
End Sub

Private Sub Command2_Click()
 Unload FrmObj
 ' Reclaim memory.
 Set FrmObj = Nothing
End Sub
```

มาถึงจุดนี้ผู้เขียนก็ขออนุญาตพักยกชั่วคราวครับ เพราะรู้สึกว่าจำนวนหน้าเริ่มที่จะมากพอสมควรแล้ว ในส่วนของเทคนิคการออปติไมซ์ที่เหลือก็จะขอยกยอดไปเขียนกันต่อในตอนต่อไปครับ และในตอนสุดท้ายของบทความหัวข้อนี้ก็จะแสดงตัวอย่างวิธีการทดสอบความเร็วในการประมวลผลของโค้ดโดยใช้ฟังก์ชันจากไฟล์ไลบรารี Benmrk32.dll ช่วยในการจับเวลาในการประมวลผลของโค้ดในระดับของมิลลิวินาที ซึ่งผู้อ่านสามารถนำไฟล์ไลบรารี Benmrk32.dll ไปใช้เป็นเครื่องมือในการพัฒนาโค้ดได้ฟรีครับ พิเศษครับสำหรับผู้อ่านที่ไม่นิยมการรอก็ให้ส่งอี-เมล์มายังแอดเดรส Sutthisakp@yahoo.com ครับ ผู้เขียนจะได้จัดส่งกลับทางอี-เมล์ฟรีครับ แต่ถ้าหากรอได้ก็สั่งซื้อได้ทางซอฟต์เมล์ของวารสารไมโครฯครับ สวัสดีครับ

เทคนิคประจำคอลัมน์

ถาม Visual Basic for Application และ Visual Basic 5.0 มีความสำคัญกันอย่างไร ?

ตอบ Visual Basic for Application เป็นภาษาประเภทสคริปต์ที่ถูกสร้างขึ้นมาเพื่อแทนที่เทคโนโลยีแมคโคร (macro) ของแอพพลิเคชันในชุดไมโครซอฟต์ออฟฟิส เช่น Microsoft Word หรือ Microsoft Excel เป็นต้น ซึ่งถือว่าภาษาประเภทสคริปต์ที่ช่วยให้การแชร์โค้ดระหว่างแอพพลิเคชันเป็นไปโดยมีมาตรฐานมากขึ้น และที่สำคัญโค้ด Visual Basic for Application ยังคอมแพตทิเบิลกับ Visual Basic อีกด้วย โดยไมโครซอฟต์ได้กำหนดให้ Visual Basic for Application เป็นส่วนหนึ่งของ Visual Basic

Visual Basic for Application จะถูกจัดเก็บเอาไว้ในไฟล์ไลบรารี Vba5.dll ซึ่งผู้อ่านสามารถเชื่อมต่อเข้ากับ Visual Basic ได้โดยเลือก Visual Basic for Application ภายในไดอะล็อกบ็อกซ์ Reference และสามารถค้นหาองค์ประกอบทั้งหมดของ Visual Basic for Application ได้โดยอาศัย Object Browser